

Soluzioni Esami SML

Giugno 2015

```
datatype naturale = zero | successivo of naturale;

val rec somma = fn
  zero      => (fn n => n)
  | successivo a => (fn n => successivo (somma a n));
```

```
val rec prodotto = fn zero      => (fn n => zero)
  | n1              => (fn zero => zero)
  | successivo n2 => if (n2 = zero) then
    1
  else
    prodotto(somma n1 n1) (n2));
```

Luglio 2015

```
datatype Expr = X
  | Y
  | Avg of Expr * Expr
  | Mul of Expr * Expr;
```

```
val rec compute = fn X      => (fn x => fn y => x)
  | Y                      => (fn x => fn y => y)
  | Avg(e1, e2) => (fn x => fn y => ((compute e1 x y) + (compute e2 x y)) div 2)
  | Mul(e1, e2) => (fn x => fn y => (compute e1 x y) * (compute e2 x y))
```

Agosto 2015

```
val rec elementi_pari = fn []      => []
  | [v]                => []
  | [a,b]              => [b]
  | a::(b::l)          => b::(elementi_pari l);
```

Settembre 2015

```
datatype codice = rosso of string
  | giallo of string
  | verde of string;
```

```
val rec arriva = fn
  []      => (fn x      => [x])
  | (verde n)::l => ((fn (verde nn) => (verde n)::(arriva l (verde nn))
    | x      => x::(verde n)::l))
  | (giallo n)::l => (fn (verde nn) => (giallo n)::(arriva l (verde nn))
    | (giallo nn) => (giallo n)::(arriva l (giallo nn))
    | x      => x::(giallo n)::l))
  | (rosso n)::l => (fn x => (rosso n)::(arriva l x));
```

Giugno 2016

```
val rec hist = fn
  []      => (fn (c:real, d:real) => 0)
| [e]     => (fn (c:real, d:real) => if (e > (c-d) andalso e < (c+d)) then
                                   1
                                   else
                                   0)
| (e :: l) => (fn (c:real, d:real) => if (e > (c-d) andalso e < (c+d)) then
                                   1 + hist l (c, d)
                                   else
                                   0 + hist l (c, d));

val rec noduplen = fn []      => 0
                  | [a]      => 1
                  | a::(b::l) => if (a <> b) then
                                1 + noduplen (b::l)
                                else
                                0 + noduplen (b::l);
```

Luglio 2016

```
datatype lambda_expr = Var of string
                    | Lambda of string * lambda_expr
                    | Apply of lambda_expr * lambda_expr;
```

```
val rec is_free =
  fn s => fn Var v => s = v
        | Lambda (v, e) => if (s = v) then
                            false
                            else
                            is_free s e
        | Apply (e1, e2) => (is_free s e1) orelse (is_free s e2);
```

```
local
  val rec eval = fn costante    n      => n
                  | somma      (a1, a2) => (eval a1) + (eval a2)
                  | sottrazione (a1, a2) => (eval a1) - (eval a2)
                  | prodotto    (a1, a2) => (eval a1) * (eval a2)
                  | divisione   (a1, a2) => (eval a1) div (eval a2);
in
  val semplifica = fn costante    n      => costante(n)
                    | somma      (a1, a2) => costante((eval a1) + (eval a2))
                    | sottrazione (a1, a2) => costante((eval a1) - (eval a2))
                    | prodotto    (a1, a2) => costante((eval a1) * (eval a2))
                    | divisione   (a1, a2) => costante((eval a1) div (eval a2))
end;
```

Agosto 2016

```
type insiemediinteri = int -> bool;

val vuoto:insiemediinteri = fn n => false;

val aggiungi = fn f:insiemediinteri => fn x:int =>
  (fn n:int => if (n = x)
    then
      true
    else
      f n
  ):insiemediinteri;

val contiene = fn f:insiemediinteri => fn n:int => f n;
```

```
val intersezione = fn i1:insiemediinteri => fn i2:insiemediinteri =>
  (fn n =>
    ((contiene i1 n) andalso (contiene i2 n))
  ):insiemediinteri;
```

Febbraio 2017

```
val unione = fn i1:insiemediinteri => fn i2:insiemediinteri =>
  (fn n =>
    ((contiene i1 n) orelse (contiene i2 n))
  ):insiemediinteri;
```

Giugno 2017

```
val rec sommali = fn z => fn []      => z
                      | v::[]      => z
                      | v1::v2::l => v2 + (sommali z l)



---



val rec sommali = fn z => fn []      => z
                      | v::[]      => z
                      | v1::v2::[]  => z
                      | v1::v2::v3::l => v3 + (sommali z l)
```

Luglio 2017

```
val rec eval = fn For (n, f) =>
  fn x => if (n > 0) then
    eval (For (n - 1, f)) (f x)
  else
    x;



---



val rec eval = fn For (n, f) =>
  fn x => if (n > 1) then
    eval (For (n - 1, f)) (f x)
  else
    x;
```

Settembre 2017

```
val lega = fn e:ambiente =>
  fn nome =>
    fn valore =>
      (fn n => if (n = nome)
        then
          (Int valore)
        else
          (e n)): ambiente;
```

Gennaio 2018

```
datatype lambda_expr = Var of string
                        | Lambda of string * lambda_expr
                        | Apply of lambda_expr * lambda_expr;
```

```
val rec is_bound =
  fn s => fn Var v => s = v
        | Lambda (v, e) => if (s = v) then
            true
            else
                is_bound s e
        | Apply (e1, e2) => (is_bound s e1) orelse (is_bound s e2);
```

Giugno 2018

```
val rec conta = fn []      => 0
                | a::l     => if (List.exists (fn n => n = a) l) then
                    (conta l)
                    else
                        1 + conta l;
```

Luglio 2018

```
datatype ITER = Iter of int * (int -> int);

val rec eval = fn Iter (n, f) =>
  fn x => if (n > 0) then
    eval (Iter (n - 1, f)) (f x)
  else
    x;
```